VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Václav Menšík

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SOFTWAROVÝ ZVUKOMĚR
SOFTWARE SOUND LEVEL METER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Václav Menšík**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jiří Schimmel, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Václav Menšík ***ID:*** 203740
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Softwarový zvukoměr

POKYNY PRO VYPRACOVÁNÍ:

Pomocí frameworku Juce nebo jiného vytvořte aplikaci pro dvoukanálové měření elektrického napětí a jeho hladiny a hladiny akustického tlaku pomocí zvukové karty. Aplikace bude umožňovat použití kmitočtových váhových filtrů A, C a Z a integračního článku s časovou konstantou Fast a Slow podle doporučení IEC 61672. Aplikaci bude možné kalibrovat pomocí známé hodnoty napětí nebo akustického tlaku na vstupu, aby zobrazovala skutečné hladiny napětí a akustického tlaku v dBV, dBU a dB(SPL). Implementujte také plovoucí průměrování změřených hodnot. Ve stejné aplikaci, ale jako samostatné okno, implementujte jednoduchý dvoukanálový generátor harmonického a šumového signálu s možností kalibrace napětí na výstupu zvukové karty. Pro nastavování hodnot generátoru implementujte klávesové zkratky.

DOPORUČENÁ LITERATURA:

[1] IEC 61672 Electroacoustics - Sound level meters.

[2] KUO, S., M.; LEE, B., H.; TIAN, W. Real-time digital signal processing: implementations and applications. 2nd ed. Hoboken, NJ: John Wiley, 2006. ISBN 0-470-01495-4.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá zařízeními na měření hluku, návrhem a realizací softwarového zvukoměru. Jsou v ní popsány vybrané akustické veličiny a podmínky kladené na zvukoměry standardem IEC 61672. Cílem této práce je implementace programu SoundMeter, jeho uživatelského rozhraní a funkcí pro zpracování signálu.

KLÍČOVÁ SLOVA

Zvukoměr, MATLAB, C++, JUCE, DSP, SPL, hladina akustického tlaku

ABSTRACT

This thesis discusses noise measurement devices, design and implementation of a software sound pressure meter. Sound pressure meter requirements specified by the IEC 61672 standard and selected acoustic physical properties are discussed. Further, the JUCE framework is presented along with an of the graphical user interface and digital signal processing capabilities of the SoundMeter program.

KEYWORDS

Sound level meter, MATLAB, C++, JUCE, DSP, SPL, sound pressure level

Vysázeno pomocí balíčku `thesis` verze 4.03; `http://latex.feec.vutbr.cz`

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Václav Menšík
VUT ID autora:	203740
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Softwarový zvukoměr

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Jiřímu Schimmelovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod **9**

1 Zvuk a měřené veličiny **10**
1.1 Akustický tlak . . . 10
1.1.1 Decibel . . . 10
1.1.2 Hladina akustického tlaku . . . 10
1.2 Převod na elektrický signál . . . 11
1.2.1 Tlakový mikrofon . . . 11
1.2.2 Mikrofony pro použití ve volném poli . . . 11
1.2.3 Mikrofony pro použití v difuzním poli . . . 11
1.2.4 Signálový řetězec . . . 12
1.3 Digitální reprezentace zvukového signálu . . . 12
1.3.1 Analogově-digitální převod . . . 13
1.3.2 Externí zvuková rozhraní k počítači . . . 13

2 Funkce zvukoměru a standard IEC 61672 **15**
2.1 Požadavky na zvukoměr . . . 16
2.2 Frekvenční váhování signálu . . . 16
2.2.1 Váhování Z . . . 17
2.2.2 Váhování C . . . 17
2.2.3 Váhování A . . . 17
2.3 Časové váhování signálu . . . 18

3 Použité softwarové nástroje **19**
3.1 Prostředí MATLAB . . . 19
3.2 Jazyk C++ . . . 19
3.3 REAPER . . . 19
3.4 Framework JUCE . . . 19
3.4.1 Projucer . . . 20
3.4.2 Grafické rozhraní . . . 20
3.4.3 Základní zpracování signálu . . . 20
3.4.4 Třída AudioProcessor . . . 21
3.4.5 Třída AudioProcessorGraph . . . 22
3.4.6 Modul dsp . . . 22

4 Program SoundMeter **26**
4.1 Zpracování signálu . . . 26
4.1.1 Zpracování vstupu a výstupu . . . 26

4.1.2 Třída ConversionProcessor 27
4.1.3 Implementace frekvenčního váhování 27
4.1.4 Implementace časového váhování 28
4.1.5 Implementace kalibrace zvukoměru 29
4.1.6 MeasurementProcessor 30
4.1.7 Signálový generátor 32
4.2 Práce s daty a stavem aplikace 33
4.3 Grafické rozhraní 35
4.3.1 Komponenta ChannelComponent 35
4.3.2 Komponenta MeterComponent 35
4.3.3 Komponenta SignalGeneratorComponent 36
4.4 Ověření funkčnosti programu 37
4.4.1 Funkce měření napětí 37
4.4.2 Funkce měření akustického tlaku 38

Závěr **42**

Literatura **43**

Seznam symbolů a zkratek **45**

A Obsah přiloženého archivu **46**

Seznam obrázků

1.1 Měřící mikrofon Behringer ECM8000 12
2.1 Digitální zvukoměr SVAN 979 15
2.2 Fletcherovy-Munsonovy křivky 16
2.3 Graf frekvenční odezvy filtru C vygenerovaného v prostředí MATLAB 17
2.4 Graf frekvenční odezvy váhovacího filtru A vygenerovaného v prostředí MATLAB 18
3.1 Ukázka rozmisťování komponent v JUCE aplikaci 21
3.2 Grafická reprezentace sériového zpracování signálu pomocí AudioProcessorGraph 23
3.3 Grafická reprezentace paralelního zpracování signálu pomocí AudioProcessorGraph 24
4.1 Spektrální analýza filtrovaného šumu - Bez zařazení filtru 28
4.2 Spektrální analýza filtrovaného šumu - Váhovací filtr C 29
4.3 Spektrální analýza filtrovaného šumu - Váhovací filtr A 30
4.4 Diagram operací nad signálem v procesoru TimeWeightingProcessor . 31
4.5 Signálový řetězec pro jeden zobrazovací prvek 31
4.6 Graf signálových cest pro vícero zobrazovacích prvků 32
4.7 Graf využití signálových cest pro výpočet jednotlivých veličin 33
4.8 Postup generování signálových cest pro MeasurementProcessor 34
4.9 Hlavní okno programu SoundMeter 36
4.10 Okno signálového generátoru v programu SoundMeter 37
4.11 Výsledek měření hladiny napětí s frekvenčním váhováním Z 38
4.12 Výsledek měření hladiny napětí s frekvenčním váhováním C 39
4.13 Výsledek měření hladiny napětí s frekvenčním váhováním A 39
4.14 Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování Z) 40
4.15 Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování A) 41
4.16 Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování C) 41

Úvod

Tato práce se věnuje návrhu a implementaci softwarového zvukoměru, jenž je možno použít na počítači se zvukovým rozhraním a měřícím mikrofonem. Tento nástroj by mohl být užitečný v prostředí, které již disponuje zvukovými rozhraními (domácí studio, audiovizuální laboratoře...), jelikož redukuje pořizovací cenu pouze na doplnění měřícího mikrofonu, oproti jednoúčelovému digitálnímu zvukoměru.

Zvukoměry jsou používány ke zjišťování hladiny akustického tlaku, a to jak v laboratorním prostředí, tak i pro objektivní posouzení, zda hlučná prostředí vyhovují hygienickým standardům (v hlučných průmyslových provozech, na koncertech).

V práci jsou popsány poznatky o zvuku, které byly při návrhu zvukoměru využity a použité softwarové nástroje a knihovny. Pro vlastní realizaci programu byl zvolen jazyk C++ a knihovna JUCE, jelikož poskytuje široké spektrum předpřipravených nástrojů, jako například zvukový vstup a výstup, grafické komponenty a moduly pro zpracování zvuku.

1 Zvuk a měřené veličiny

Zvuk je podélné vlnění šířené látkovým prostředím, v případě vzduchu měřitelné jako fluktuace v atmosferickém tlaku. Vzniká mechanickou oscilací zdroje zvuku, například kmitáním membrány reproduktoru či struny hudebního nástroje. [3]

1.1 Akustický tlak

Za přítomnosti zvuku jako akustický tlak považujeme rozdíl mezi okamžitou hodnotou celkového tlaku p_C a statickou hodnotou atmosférického tlaku p_{00}. Jde o střídavou složku tlaku superponovanou k průměrnému atmosferickému tlaku. Jednotkou tlaku je Pascal (Pa). [5]

Efektivní hodnota akustického tlaku je počítána dle vztahu 1.1. Dále v této práci p značí efektivní hodnotu akustického tlaku, okamžitá hodnota bude značena $p(t)$. [4]

$$p = \sqrt{\frac{1}{T}\int_0^T p^2(t)dt} \qquad [\text{Pa}] \quad (1.1)$$

1.1.1 Decibel

Lidské ucho vnímá akustický tlak nelineárně, za účelem vyjadřování velkých rozsahů hodnot akustických a elektrických veličin byla zavedena jednotka decibel (dB). Hladina L_x libovolné energetické veličiny x je vypočtena dle 1.2.

$$L_x = 20 \log \frac{x}{x_0} \qquad [\text{dB}] \quad (1.2)$$

1.1.2 Hladina akustického tlaku

Hladina akustického tlaku je odvozena z efektivní hodnoty akustického tlaku. Slouží k logaritmickému vyjádření efektivní hodnoty akustického tlaku vztažené k akustickému tlaku $p_0 = 2 \cdot 10^{-5}$ Pa odpovídajícímu prahu slyšení při kmitočtu 1 kHz. Jako jednotku uvádíme dB (SPL). [5]

$$L_p = 10 \log \frac{p^2}{p_0^2} = 20 \log \frac{p}{p_0} \qquad [\text{dB(SPL)}] \quad (1.3)$$

1.2 Převod na elektrický signál

Za účelem měření či dalšího zpracování zvuku je potřeba jej převést do podoby elektrického signálu. K tomuto slouží mikrofony, které díky pohybu membrány způsobenému akustickým tlakem vytvářejí změny elektrického napětí či proudu. Pro záznam zvuku jsou široce využívané mikrofony dynamické a kapacitní, případně piezoelektrické. Pro účely této práce jsou zmíněny druhy kapacitních mikrofonů užívané k akustickému měření.

Vlastnost mikrofonů, která vyjadřuje vztah mezi akustickým tlakem působícím na membránu a výstupním napětím mikrofonu, nazýváme *citlivost* a vyjadřujeme ji v milivoltech na pascal. [3] Na měřící mikrofony je kladen požadavek vyrovnané frekvenční charakteristiky a kulové směrové charakteristiky. Výrobci meřících mikrofonů k nim také dodávají graf frekvenční a směrové charakteristiky, případně kalibrační soubor.

1.2.1 Tlakový mikrofon

Tlakový mikrofon reaguje na změny akustického tlaku pouze jednou membránou, ideální tlakový mikrofon není ovlivněn směrovou charakteristikou. Při dopadu vlny kolmo na membránu dochází ke zvýšení tlaku na membráně, jelikož samotný mikrofon je překážkou v šíření zvuku. Mikrofony mohou být vůči tomuto jevu kompenzovány úpravou konstrukce, zejména ochranné mřížky před membránou. Reálné tlakové mikrofony tedy vykazují směrovost.[16]

Dalším z faktorů je při vysokých frekvencích zvlnění membrány, když se vlnové délky vln (dopadajících ze směru mimo osu membrány) blíží průměru membrány. [3]

1.2.2 Mikrofony pro použití ve volném poli

Mikrofony pro použití ve volném poli, anglicky nazývané *free-field* svou konstrukcí kompenzují problém tlakových mikrofonů, kde je měření akustického tlaku ovlivněno tím, že samotný mikrofon je překážkou v šíření zvuku. Tyto mikrofony je, jak název napovídá, vhodno používat v případě, že v prostředí měření nejsou přítomny odrazivé plochy (např. bezodrazové komory, otevřený prostor). Tyto mikrofony je třeba orientovat ve směru zdroje zvuku.

1.2.3 Mikrofony pro použití v difuzním poli

Mikrofony pro použití v difuzním poli, anglicky nazývané *random-incidence* nebo *diffusion-field*, jsou kompenzovány pro ideální kulovou charakteristiku. Tento typ

Obr. 1.1: Měřící mikrofon Behringer ECM8000.[12]

mikrofonů je vhodno používat při měření s více zvukovými zdroji, v prostorech s velkými odrazivými plochami a obecně v uzavřených prostorech. [15]

1.2.4 Signálový řetězec

Výstup mikrofonu je pro záznam či měření nutno přivést na předzesilovač. Pro účely využití softwarového zvukoměru se předpokládá využití zvukového rozhraní, které na každém vstupu poskytuje integrovaný mikrofonní předzesilovač, phantomové napájení +48 V pro kapacitní mikrofony a analogově-digitální převodník.

Předzesilovač poskytuje nastavitelné zesílení vstupního signálu. Při volbě zesílení je třeba nastavit rozsah tak, aby signál nebyl přebuzen (indikátor CLIP, OVERLOAD na audio rozhraní či samostatném předzesilovači). Přebuzení do signálu vnáší zkreslení, znehodnocuje tedy snahy měření. Součástí předzesilovače také bývá ovládací prvek PAD, jenž zařazuje před předzesilovač útlumový článek (obvykle útlum -10 či -20 dB), a může být užitečný při použití velmi citlivého mikrofonu či záznamu vysokého akustického tlaku.

Za předzesilovačem následuje analogově-digitální převodník, jenž zajišťuje kvantizaci a vzorkování signálu (viz 1.3.1).

1.3 Digitální reprezentace zvukového signálu

Digitální doména nedovoluje uchovávat reálné signály ve spojité podobě. Tento problém řeší pulzně kódová modulace (PCM - Pulse-code modulation). Časová osa

spojitého signálu je tedy nahrazena diskrétní osou a vzorkovacím kmitočtem, který určuje, kolik vzorků diskrétní osy odpovídá jedné sekundě spojité osy. Hodnota signálu je reprezentována jako číslo v omezeném rozsahu a s omezeným rozlišením. [1] Tento rozsah a rozlišení jsou udány datovým typem použitým pro ukládání jednotlivých hodnot. Například při použití 16 bitového celého čísla bez znaménka (16 bit unsigned integer) může signál nabývat hodnot od 0 do 2^{16}, přičemž 0 odpovídá maximální záporné výchylce a 2^{16} maximální kladné výchylce.

1.3.1 Analogově-digitální převod

Pro převádění analogového signálu na digitální využíváme operací vzorkování a kvantizace. Vzorkování je prováděno obvodem Sample & Hold, jenž ze vstupu získá vstupní napětí a udržuje jej na výstupu konstantní po dobu vzorkovací periody T_{vz} získanou dle 1.4.

$$T_{vz} = \frac{1}{f_{vz}} \tag{1.4}$$

Navzorkovaný signál dále přechází ke kvantizaci, při níž je k hodnotě napětí signálu přiřazována digitální hodnota. A/D převodník má vlastnost zvanou rozlišení, která vyjadřuje počet kvantovacích úrovní. Jelikož v digitální doméně pracujeme s omezenou přesností, je nutno k převodu na digitální hodnotu využít nejbližší kvantovací úroveň napětí. Rozdíl mezi spojitým vstupním signálem a kvantovaným signálem pak nazýváme kvantovací chyba. [1]

1.3.2 Externí zvuková rozhraní k počítači

Pro účely této práce jsou diskutována pouze dedikovaná zvuková rozhraní, jelikož integrovaná zařízení v základních deskách počítačů a laptopech jsou kvůli absenci XLR vstupu, phantomového napájení a nízkému odstupu signálu od šumu nevhodná pro měřící aplikace.

Dedikovaná zvuková rozhraní jsou k počítači připojována pomocí jedné z následujících sběrnic:

- USB - Universal Serial Bus
- FireWire
- Thunderbolt
- PCI-Express
- PCI

Pro použití takovéhoto zařízení je třeba na počítači nainstalovat *driver* poskytnutý výrobcem, který bude obstarávat komunikaci mezi softwarem běžícím na počítači a samotným zařízením. Některá zařízení jsou navržena jako *plug'n'play*, neměla by tedy požadovat instalaci driveru, namísto toho využívat univerzální driver zabudovaný v operačním systému (např. na platformě Mac, iPad, Linux). V případě Windows ke studiovým zvukovým rozhraním výrobci driver dodávají, především kvůli podpoře technologie ASIO, která na platformě Windows obchází vestavěný audio mixér a zajišťuje komunikaci s audio rozhraním s nižšími prodlevami.

Přenos signálu mezi zvukovým rozhraním a počítačem je realizován v blocích vzorků, tzv. *vyrovnávací paměti*. Tyto bloky vyrovnávací paměti jsou z rozhraní odeslány do počítače, kde ovladač příchozí vyrovnávací paměť předá do aplikace, která zařízení používá. Aplikace vstupní vyrovnávací paměť zpracuje a připraví blok výstupní vyrovnávací paměti, která je předána ovladači a odeslána na zvukové rozhraní. Velikost vyrovnávací paměti je spolu s vzorkovací frekvencí možno nastavit v ASIO driveru, v případě platformy Mac pak přímo v aplikaci.[14]

2 Funkce zvukoměru a standard IEC 61672

Zvukoměr je zařízení sloužící k měření hladiny akustického tlaku. Zvukoměry jsou využívány při laboratorním měření akustických veličin či měření za účelem potvrzení, že stroje nebo přístroje splňují zákonné normy na emise hluku. Další z využití zvukoměrů jsou při přípravě a průběhu hudební nebo audiovizuální produkce pro objektivní zajištění dodržování hygienických norem, případně potvrzení jejich nedodržení. [19] [20]

Zvukoměry bývají používány i při sběru dat pro hlukové studie industriálních provozů a studie zabývající se hlukovým znečištěním životního prostředí a jeho vlivy na lidské zdraví.

Moderní implementace zvukoměrů dovolují autonomní provoz, přičemž měřená data jsou odesílána na server, odkud je možno je stáhnout, či monitorovat aktuální odečty. Toto může sloužit pro krátkodobá měření na více lokacích nebo dlouhodobou instalaci pro sběr a monitoring dat. Například řešení NoiseScout od společnosti NTi Audio poskytuje i možnost připojení meteorologické stanice po boku zvukoměru pro záznam vlivů počasí, které mohou mít vliv na výsledky měření hluku. [18]

Obr. 2.1: Digitální zvukoměr SVAN 979.[13]

2.1 Požadavky na zvukoměr

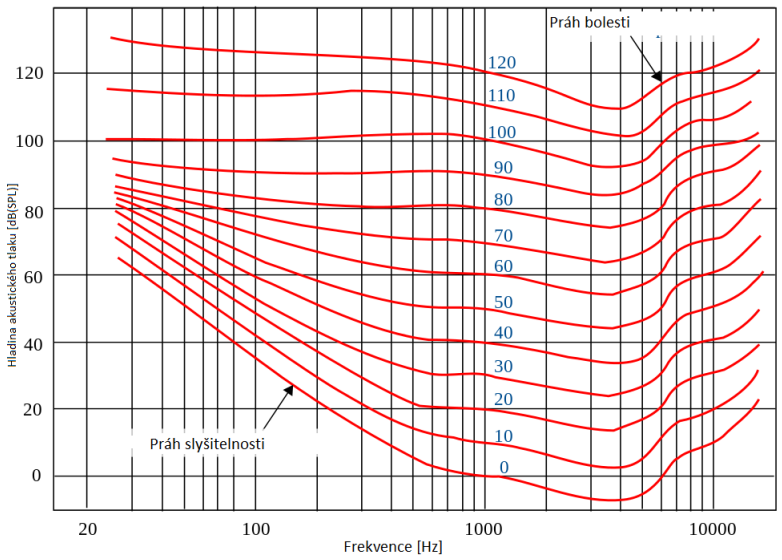
Standard IEC 61672 uvádí, že zvukoměr sestává z mikrofonu, signálového procesoru a zobrazovacího zařízení. Za signálový procesor se považuje zařízení schopné provádět výpočet čtverců hodnot frekvenčně váhovaných okamžitých hodnot akustického tlaku a jejich integraci či průměrování v čase.

Na zobrazovací zařízení zvukoměru je kladen požadavek, že musí poskytovat fyzický displej či úložiště, přičemž uložené výsledky měření musí být přístupné pomocí zařízení určeného výrobcem, například připojením k počítači s příslušným softwarem. [2]

Pro účely této práce je u softwarového zvukoměru za signálový procesor považován počítač s audio rozhraním a za zobrazovací zařízení uživatelské rozhraní programu.

2.2 Frekvenční váhování signálu

Lidské vnímání hlasitosti je závislé na frekvenci. Vztah vnímané hlasitosti a frekvence a hladiny akustického tlaku je popsán tzv. Fletcherovými-Munsonovými křivkami (viz 2.2). Každá z křivek spojuje body grafu odpovídající stejné vnímané hlasitosti. Váhování signálu slouží při měření akustických veličin k aproximaci frekvenční závislosti citlivosti lidského ucha.



Obr. 2.2: Fletcherovy-Munsonovy křivky stejné hlasitosti, přeloženo z angličtiny. [11]

Norma IEC 61672 specifikuje frekvenční odezvu jednotlivých váhování pomocí závislosti zesílení ($Z(f)$, $C(f)$, $A(f)$) na frekvenci f. [2]

2.2.1 Váhování Z

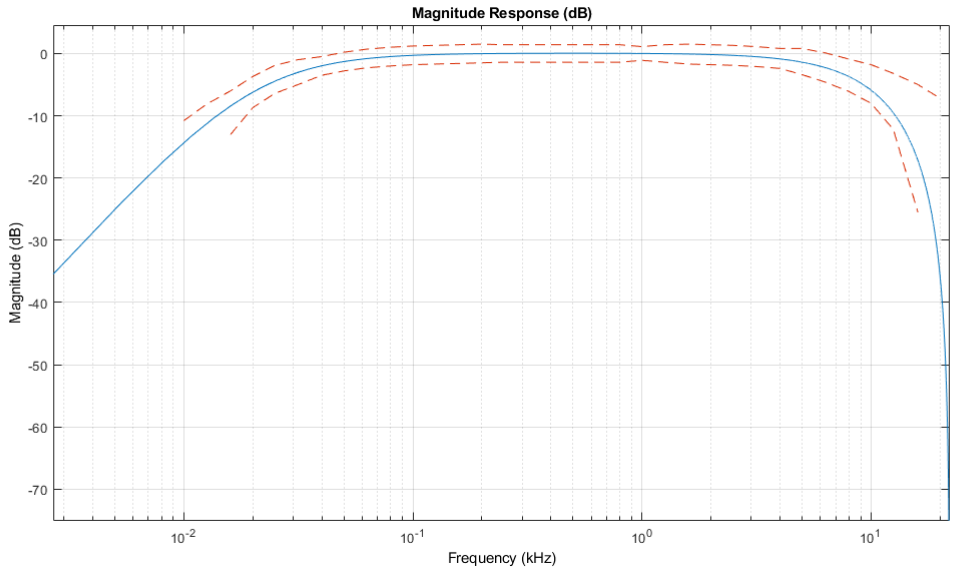
Váhování Z je definováno jako zesílení 0 dB pro libovolnou frekvenci f, viz 2.1. Ačkoli v ideálním stavu vůbec signál nemění, jsou pro něj definovány tolerance na jednotlivých frekvencích, které implementace musí splňovat. [2]

$$Z(f) = 0 \qquad [\text{dB}] \quad (2.1)$$

2.2.2 Váhování C

Závislost zesílení váhování C na frekvenci f je definován podle vztahu 2.2, kde f_1 je přibližně 20,6 Hz a f_4 přibližně 107,7 Hz. C_{1000} je normalizační konstanta, která ve výsledné závislosti zajišťuje zesílení 0 dB při frekvenci 1000 Hz. [2]

$$C(f) = 20 \log \frac{f_4^2 f^2}{(f^2 + f_1^2)(f^2 + f_4^2)} - C_{1000} \qquad [\text{dB}] \quad (2.2)$$



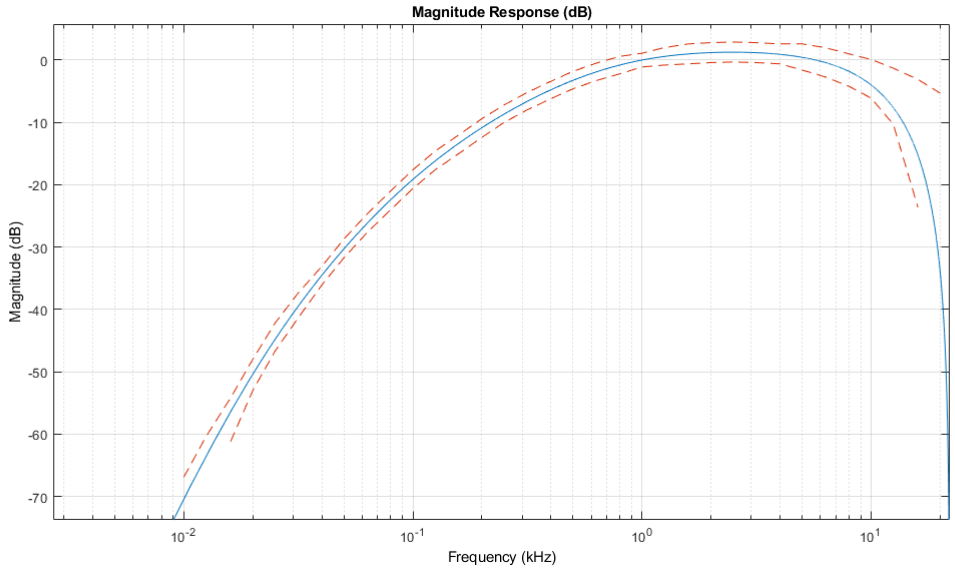
Obr. 2.3: Graf frekvenční odezvy filtru C vygenerovaného v prostředí MATLAB.

2.2.3 Váhování A

Závislost zesílení váhování A na frekvenci f je dáno vztahem 2.3, kde f_1 a f_4 jsou shodné s hodnotami u váhování C, f_3 je přibližně 737,9 Hz a f_4 je přibližně 12194 Hz.

A_{1000} je normalizační konstanta, která ve výsledné závislosti zajišťuje zesílení 0 dB při frekvenci 1000 Hz. [2]

$$A(f) = 20 \log \frac{f_4^2 f^4}{(f^2 + f_1^2)(f^2 + f_4^2)} - A_{1000} \qquad \text{[dB]} \quad (2.3)$$



Obr. 2.4: Graf frekvenční odezvy filtru A vygenerovaného v prostředí MATLAB.

2.3 Časové váhování signálu

Pro snížení vlivu krátkodobých výkyvů napětí či akustického tlaku na přesnost odečítání měřených hodnot ze zobrazovacího zařízení je využito časové váhování pomocí integračního článku definované vztahem 2.4, kde τ je konstanta časového váhování a ξ je pomocná proměnná. [2]

$$Ł_{A_\tau}(t) = 20 \log \frac{\sqrt{\frac{1}{\tau} \int_{-\inf}^{t} p_A^2(\xi) e^{-(t-\xi)/\tau} d\xi}}{p_0} \qquad (2.4)$$

3 Použité softwarové nástroje

3.1 Prostředí MATLAB

Prostředí MATLAB od společnosti Mathworks bylo využito při prototypování některých dílčích částí programu a také pro výpočet koeficientů váhovacích filtrů A a C. K tomuto sloužil DSP System Toolbox, který obsahuje nástroje k návrhu filtrů, včetně specifických funkcí pro filtry váhovací.

3.2 Jazyk C++

Zpracování audio signálu v reálném čase je citlivé na výpočetní rychlost, jazyk C++ je pro toto vhodný, jelikož vývojářům dovoluje nízkoúrovňovou a efektivní práci se signálem. Jeho podpora objektově orientovaného programování dovoluje zaobalovat části kódu pro přehledné a snadné opakované využití. Naříklad použitá knihovna JUCE díky paradigmatu OOP využívá vysokou míru abstrakce, vývojář se tedy může soustředit především na vlastní zpracování signálu a stavby uživatelského rozhraní pomocí předpřipravených tříd a funkcí.

3.3 REAPER

Program REAPER vyvíjený společností Cockos Inc. je DAW (digital audio workstation), tedy software určený pro nahrávání, střih a zpracování zvuku a MIDI dat. Podporuje použití VST zásuvných modulů a také poskytuje různé signálové generátory a frekvenční analyzátor. Pro účely této práce byl program použit k ověření správného fungování váhovacích filtrů, viz 4.1.3.

3.4 Framework JUCE

Framework JUCE slouží pro vývoj zvukového software (samostatných programů a plug-in modulů). Zahrnuje možnosti tvorby grafického uživatelského rozhraní, správu vstupně-výstupních toků signálu, komunikaci se zvukovými rozhraními, moduly pro zpracování signálu a další funkcionalitu. Knihovna byla poprvé vydána v roce 2004 Julianem Storerem, nyní vývoj pokračuje pod společností ROLI. JUCE podporuje mimo operační systémy Windows a Mac také vývoj pro Linux, iOS a Android. Multiplatformně navržená knihovna poskytuje možnost sdílení kódu mezi verzemi programu pro různé platformy.

3.4.1 Projucer

Spolu s použitím frameworku JUCE souvisí také program Projucer sloužící ke správě projektu, instalaci modulů knihoven JUCE a poskytnutí nastavení pro správnou kompilaci programu. Je možné jej také používat jako editor, pro tuto práci bylo k samotnému psaní, sestavování a ladění kódu použito prostředí Visual Studio 2019 a XCode pro macOS. Projucer přináší zjednodušení procesu sestavení a ladění poskytnutím přednastaveného projektu pro zvolené vývojové prostředí. Díky tomuto je možné snadno přenášet kód mezi operačními systémy bez potřeby ruční tvorby separátních projektů pro každé vývojové prostředí.

3.4.2 Grafické rozhraní

JUCE používá k tvorbě grafického rozhraní takzvané komponenty a jejich hierarchii. Celý obsah aplikace bývá obvykle umístěn v jedné hlavní komponentě, složený z dílčích částí. Knihovna JUCE poskytuje třídu `Rectangle`, která je určena k práci s plochami v okně programu. Například použití její metody `removeFromLeft(x)` oddělí část původní plochy o šířce `x` a vrátí ji k dalšímu použití. Volání zmíněné funkce přepíše také původní `Rectangle` tak, aby již neobsahovala oddělenou část. Díky tomuto systému je možné snadno rozmisťovat obsah v okně programu. Obrázek 3.1 ukazuje možné rozložení uživatelského rozhraní pomocí komponent. [7]

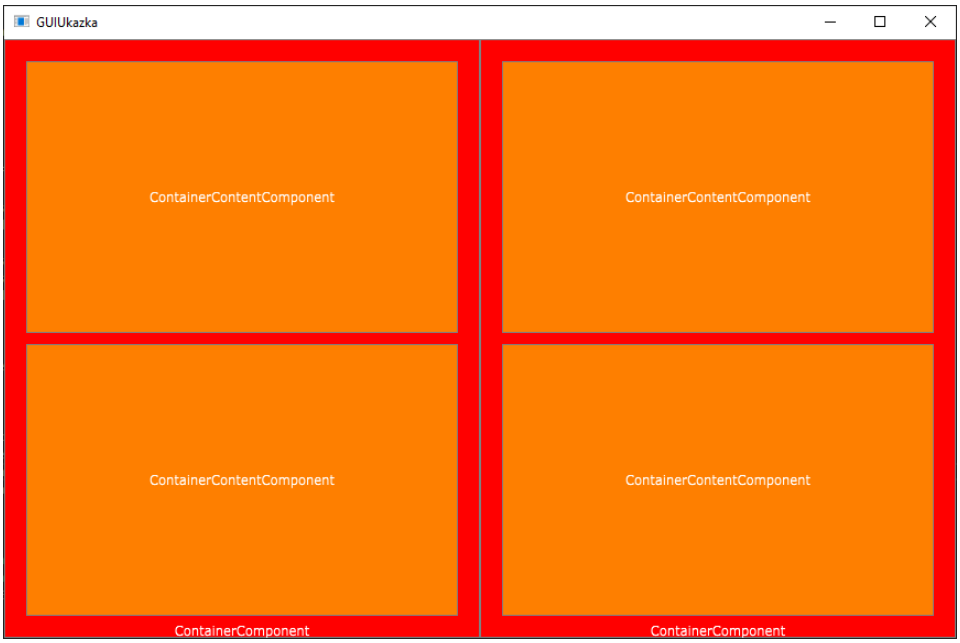
Pro vykreslování komponent v okně slouží metoda `paint`, pomocí které je možno například obarvit pozadí komponenty, vykreslovat obrazce či zobrazovat text. Grafické rozhraní je ve většině případů vykreslováno asynchronně, má nižší prioritu, než samotné zpracování signálu.

3.4.3 Základní zpracování signálu

Framework JUCE poskytuje pro tvorbu samostatných zvukových aplikací komponentu AudioAppComponent, která zahrnuje abstrakci pro komunikaci s driverem zvukového rozhraní a zjednodušuje základní nastavení zvukového zařízení a přístup ke vstupním a výstupním signálů do pouhých několika funkcí. Pro tuto práci byly důležité následující.

Funkce `prepareToPlay` je volána při spuštění programu či změně zvukového rozhraní, dovoluje předat programu informace o vzorkovací frekvenci a délce vstupně-výstupní vyrovnávací paměti.

Funkce `getNextAudioBlock` poskytuje vícekanálovou vyrovnávací paměť naplněnou daty vstupních signálů. Stejný blok vyrovnávací paměti je pak po zpracování vracen zvukovému rozhraní. V rámci této funkce je možno nad vstupními daty provádět výpočty, manipulovat s daty, či je přepsat (například šumem, generovaným



Obr. 3.1: Ukázka rozmisťování komponent v JUCE aplikaci.

signálem nebo nulovými hodnotami). Tato funkce je volána při každém novém příchozím bloku vyrovnávací paměti ze zvukového rozhraní.

Na zpracování signálu v rámci funkce `getNextAudioBlock` je kladen požadavek, že musí dokončit svůj běh do doby, kdy audio rozhraní poskytne další blok vyrovnávací paměti ke zpracování. Například při vzorkovací frekvenci 44,1 kHz a délce vyrovnávací paměti 256 vzorků tedy obsah funkce musí být vykonán přibližně do 5 ms. Při nesplnění této podmínky by v signálu vznikaly nespojitosti, které se projevují jako praskání, lupání a jiné znehodnocení signálu. [7]

3.4.4 Třída AudioProcessor

Třída AudioProcessor je především užívána při tvorbě zásuvných modulů pro DAW jakožto hlavní blok kódu, v kterém probíhá zpracování signálu. V prostředí samostatné audio aplikace i zásuvného modulu je možno tuto třídu využít pro zaobalení jednotlivých částí řetězce pro zpracování signálu namísto těžko čitelného a opakujícího se kódu v rámci metody `getNextAudioBlock`.

Instance třídy AudioProcessor stejně jako samotná AudioAppComponent poskytují metodu prepareToPlay, pomocí které je možno audio procesoru předat informaci o vzorkovací frekvenci a velikosti vyrovnávací paměti a dle těchto informací nastavit vnitřní stav (např. vygenerovat koeficienty filtru, alokovat paměť pro zpožďovací linku, atd.).

Samotné zpracování signálu probíhá v metodě processBlock, do níž je předávána reference na vyrovnávací paměť pro zvukový signál (AudioSampleBuffer) a také vyrovnávací paměť pro MIDI zprávy (MIDIBuffer).

Díky tomuto rozdělení dílčích funkcionalit je možno získat přehlednější kód se snazší údržbou (úprava je prováděna na jednom místě namísto vícera) a snadnou možností opakovaného použití kódu. Tato modularita přináší možnost dynamicky měnit pořadí jednotlivých procesorů v řetězci či jejich vypínání. Knihovna JUCE také nabízí možnost instance třídy AudioProcessor řetězit a větvit pomocí třídy AudioProcessorGraph. [9] [7]

3.4.5 Třída AudioProcessorGraph

Třída AudioProcessorGraph slouží k řetězení a větvení signálu mezi instancemi třídy AudioProcessor v podobě grafové struktury. Do těchto řetězců je možno zasahovat za běhu programu - vytvářet a rušit vrcholy grafu (samotné AudioProcessory zaobalené v instanci třídy AudioProcessorGraph::Node) a také hrany grafu (instance třídy Connection) definující spojení mezi vstupy a výstupy jednotlivých procesorů.

Samotná třída AudioProcessorGraph je odvozena od AudioProcessor, poskytuje tedy metody prepareToPlay a processBlock - instance je možno použít v rámci dalších procesorových grafů, či jako hlavní signálový procesor v aplikaci či zásuvném modulu pro aplikace DAW. [10] [7]

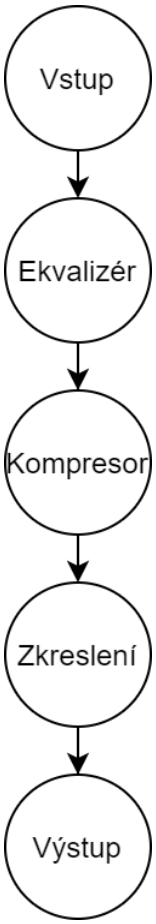
3.4.6 Modul dsp

Jeden z modulů frameworku JUCE se nazývá dsp. Jde o soubor tříd a struktur, které usnadňují zpracování signálu, v tomto modulu můžeme nalézt třídy, které poskytují základní prvky pro obecné zpracovávání a tvorbu 1D signálu (oscilátory, filtry, zpožďovací linky) i specializovanější zvukové procesory a efekty (šumová brána, dynamický kompresor, dozvuk).

Pro účely této práce jsou z modulu dsp využity třídy IIR::Coefficients, IIR::Filter pro frekvenční a časové váhování měřeného signálu a také třída Oscillator pro signálový generátor. [7]

dsp::AudioBlock

Třída AudioBlock poskytuje sjednocené rozhraní pro předávání dat ke zpracování do signálových procesorů modulu dsp z různých podob (ukazatel na pole kanálů a jejich vzorků, objekt třídy AudioBuffer). [7]



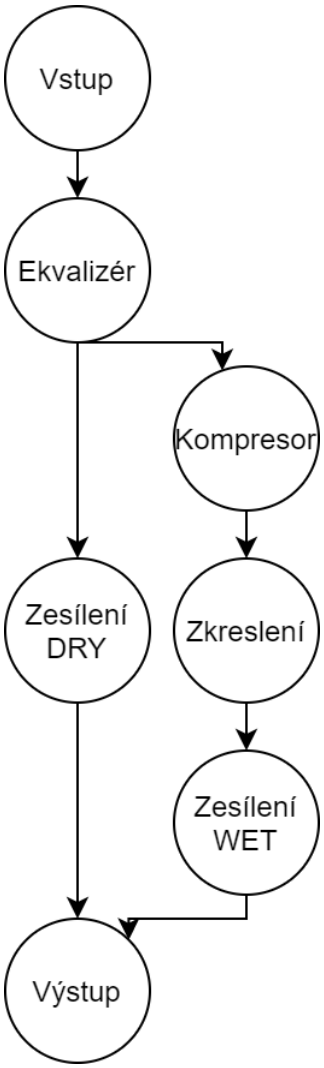
Obr. 3.2: Grafická reprezentace sériového zpracování signálu pomocí AudioProcessorGraph.

dsp::ProcessContext(Replacing/NonReplacing)

Struktura ProcessContextReplacing zaobaluje instance třídy IIR::AudioBlock spolu s informací o tom, zda zpracovaný signál má přepsat vstupní signál, či být uložen pomocí jiné instance třídy AudioBlock. V případě spojení několika signálových procesorů v sérii je vhodnou volbou ProcessContextReplacing, například při rozvětvení signálu pro paralelní zpracování je možno využít ProcessContextNonReplacing a výsledek ukládat do vyrovnávací paměti jiné než vstupní. [7]

dsp::IIR::Coefficients

Třída IIR::Coefficients z modulu dsp zaobaluje pole koeficientů pro jeden IIR filtr. Mimo samotné koeficienty zahrnuje také statické funkce, pomocí kterých je možno snadno vygenerovat sadu koeficientů pro základní filtry na základě poskytnuté vzorkovací frekvence, mezní či střední frekvence filtru a jeho řádu, popř. u peak a shelf



Obr. 3.3: Grafická reprezentace paralelního zpracování signálu pomocí AudioProcessorGraph.

filtrů nastavitelného koeficientu Q a zisku. Vývojáři je také poskytnuta možnost poskytnout předvypočtené koeficienty, v této práci jsou použity jak zabudované funkce pro generování koeficientů, tak i koeficienty vygenerované v prostředí Matlab.[7]

dsp::IIR::Filter

Třída IIR::Filter poskytuje funkcionalitu zpracování poskytnutého bloku signálu pomocí IIR filtru sestrojeného pomocí poskytnutých koeficientů. Před použitím filtru je potřeba mu předat informace o vzorkovací frekvenci, počtu kanálů a délce vyrovnávací paměti, s kterými momentálně program pracuje. Toto je provedeno pomocí tzv. ProcessSpec struktury, kterou je možno použít pro konfiguraci většiny

tříd provádejících samotné zpracování signálu z modulu dsp. O samotné zpracování signálu se stará metoda process, které je jako argument předána reference na `ProcessContextReplacing` či `ProcessContextNonReplacing`. [7]

dsp::Oscillator

Třída dsp::Oscillator slouží pro generování signálu, tvar vln tohoto signálu je volen pomocí předání ukazatele na funkci do metody `initialise`. Vstupní hodnoty do této funkce můžeme považovat za fázi oscilátoru a jsou ohraničeny $-\pi$ až $+\pi$. Po předání této funkce je vygenerována vyhledávací tabulka (Look up table), která slouží pro aproximaci funkcí náročných na výpočetní výkon. Třída `Oscillator` stejně jako `IIR:Filter` přijímá pomocí metody prepare nastavení v podobě struktury `ProcessSpec` a metoda process provádí samotné zpracování zvuku nad předanou instancí ProcessContext. [7]

4 Program SoundMeter

Výsledkem této práce je program SoundMeter, který poskytuje možnost načítání signálu ze vstupů zvukového rozhraní připojeného k počítači, zpracování tohoto signálu pomocí převodu na napětí či akustický tlak, frekvenčního a časového váhování. Pro použití softwarového zvukoměru je potřeba jej nejprve kalibrovat, tato možnost je k dispozici nezávisle u každého z měřících kanálů. Program je dále schopen zobrazovat měřené hodnoty vstupního signálu pro dva nezávislé kanály a je možno najednou měřit až šest kombinací převodu na napětí či akustický tlak, frekvenčních a časových váhování, zobrazované hodnoty jsou průměrovány (dobu průměrování volí uživatel). Program dále poskytuje možnost generování laditelného sinusového signálu či šumu na výstup zvukového rozhraní.

4.1 Zpracování signálu

Zpracování signálu je řešeno pomocí znovu použitelných bloků, tzv. procesorů, odvozených od třídy `AudioProcessor` (viz 3.4.4). Pro účely této práce jsou z této třídy důležité metody `prepareToPlay` a `processBlock`. V metodě `prepareToPlay` je prováděna inicializace filtrů (získání koeficientů a předání do příslušných instancí `dsp::IIR::Filter`) u procesorů zastávajících funkci frekvenčního a časového váhování. U procesoru pro časové váhování je také před spuštěním zpracování signálu provedena alokace vyrovnávací paměti pro účel průměrování měřených hodnot.

4.1.1 Zpracování vstupu a výstupu

Vstupní signál je v JUCE aplikaci přístupný pomocí metody `getNextAudioBlock`. Signál je zkopírován do jednotlivých jednokanálových vyrovnávacích pamětí `channel0Buffer` a `channel1Buffer` a následně předán ke zpracování měřícímu procesoru `MeasurementProcessor` příslušného kanálu. Před zpracováním zkopírovaných jednokanálových vyrovnávacích pamětí je obsah hlavní vyrovnávací paměti (`bufferToFill`) vynulován či přepsán signálem z oscilátoru (viz 4.1.7), aby vstupní signál nepřicházel na výstup a aby případné změny struktury v měřících procesorech a souvisejících realokacích paměti (viz 4.1.6) neměly za důsledek vznik praskání či jiné zvukové projevy na výstupu zvukového rozhraní. Po doběhnutí funkce `getNextAudioBlock` je objekt `bufferToFill` předán ovladači zvukového rozhraní jako blok výstupního signálu.

4.1.2 Třída ConversionProcessor

Třída `ConversionProcessor` zastává funkci převodu digitálních hodnot poskytnutých zvukovým rozhraním pomocí převodní konstanty na hodnoty odpovídající napětí na vstupu zvukového rozhraní či akustickému tlaku působícímu na připojený měřící mikrofon. Převodní konstanta je získána pomocí procesu kalibrace a kalibračního procesoru.

4.1.3 Implementace frekvenčního váhování

Související třídy ve zdrojovém kódu

- `juce::dsp::IIR::Coefficients`
- `juce::dsp::IIR::Filter`
- `WeightingFilterCoefficients`
- `FrequencyWeightingProcessor`

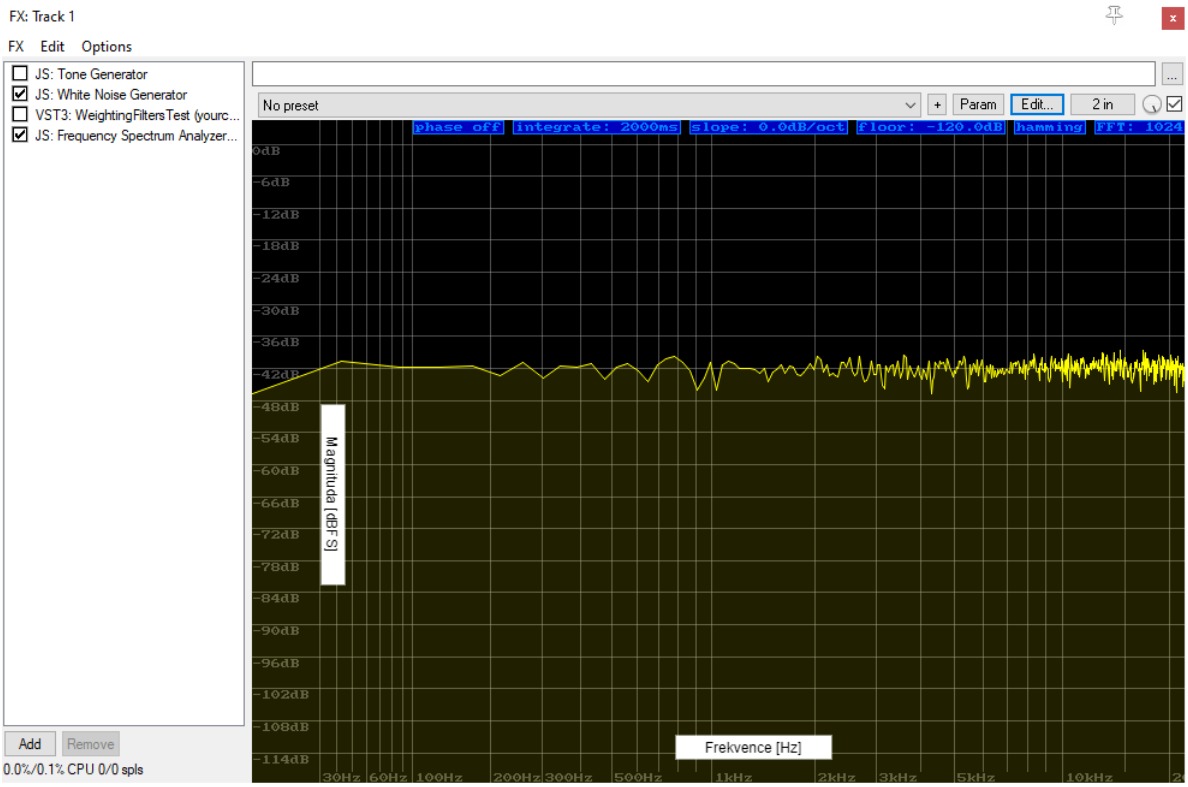
Prvotní řešení návrhu a aplikace váhovacích filtrů bylo implementováno v C++ a koeficienty filtrů byly počítány za běhu programu podle vzorkovací frekvence zvukového rozhraní. Tyto filtry však byly vysokého řádu a zvolená implementace v C++ nebyla stabilní a tedy ani vhodná, pravděpodobně kvůli náchylnosti na zaokrouhlovací chybu. Proto bylo zvoleno řešení pomocí předvypočtených koeficientů pro obvykle používané vzorkovací frekvence v audio zařízeních - 44,1 kHz, 48 kHz, 88,2 kHz a 96 kHz. [14]

Váhovací filtry byly v prostředí MATLAB navrženy pomocí objektu `fdesign.audioweighting` a pomocí funkce `sos` rozloženy na sekce druhého řádu. Koeficienty těchto filtrů pak byly použity pro vytvoření kaskády filtrů druhého řádu pomocí objektů `juce::dsp::IIR::Coefficients` a `juce::dsp::IIR::Filter` z knihovny JUCE.

Původně byla využita implementace pomocí třídy `juce::IIRCoefficients` a `juce::IIRFilter`. Třída `juce::IIRFilter` poskytuje metodu `processSamples`, která aplikuje filtr na pole zvukových vzorků předané ukazatelem. Instance této třídy uchovávají stav filtru mezi jednotlivými voláními funkce `processSamples`.

Správný průběh tohoto procesu byl ověřen vytvořením jednoduchého VST3 zásuvného modulu, jemuž byl v programu REAPER na vstup přiveden bílý šum z generátoru a výstup zásuvného modulu byl přiveden na spektrální analyzátor, viz obrázky 4.1, 4.2 a 4.3.

Frekvenční váhování C je realizováno pomocí kaskády dvou IIR filtrů druhého řádu a násobením kompenzační konstantou pro dosažení zesílení 0 dB na frekvenci 1000 Hz, frekvenční váhování A pomocí kaskády tří IIR filtrů a násobením příslušnou kompenzační konstantou.



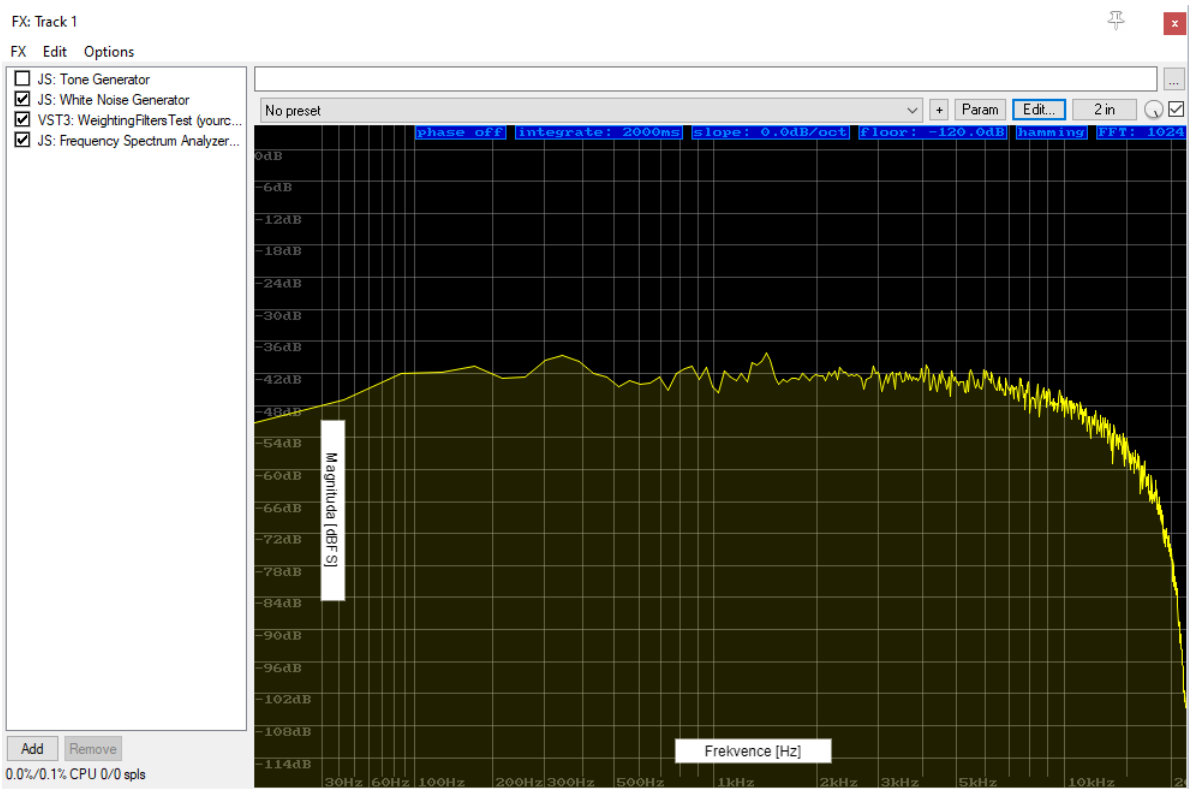
Obr. 4.1: Spektrální analýza bílého šumu - Bez zařazení filtru.

Příslušné třídy a funkce byly přepracovány do podoby procesoru `FrequencyWeightingProcessor`, a namísto třídy juce::IIRFilter využívají modernější modul `juce::dsp` (viz 3.4.6).

4.1.4 Implementace časového váhování

Časové váhování je prováděno v procesoru `TimeWeightingProcessor`, uspořádání operací prováděných na signálu je shrnuto v obrázku 4.4. Prvním krokem časového váhování je výpočet druhé mocniny jednotlivých vzorků ve vyrovnávací paměti. Toto je realizováno pomocí funkce `multiplyBy` dostupné u třídy zaobalující blok vyrovnávací paměti `juce::dsp::AudioBlock`. Oproti využití cyklu `for` tato implementace explicitně využívá tzv. SIMD instrukce, pokud jsou na vybrané cílové platformě dostupné, které dovolují zpracování většího množství vektorizovaných dat pomocí jedné instrukce, dovolují tedy výrazné urychlení této operace. Framework JUCE poskytuje mimo jiné tyto optimalizované funkce i pro kopírování, sčítání a odčítání bloků vyrovnávací paměti.

Druhým krokem je filtrace umocněného signálu pomocí dolní propusti realizované IIR filtrem prvního řádu. Mezní frekvence f_m této dolní propusti je dána vztahem



Obr. 4.2: Spektrální analýza bílého šumu - Váhovací filtr C.

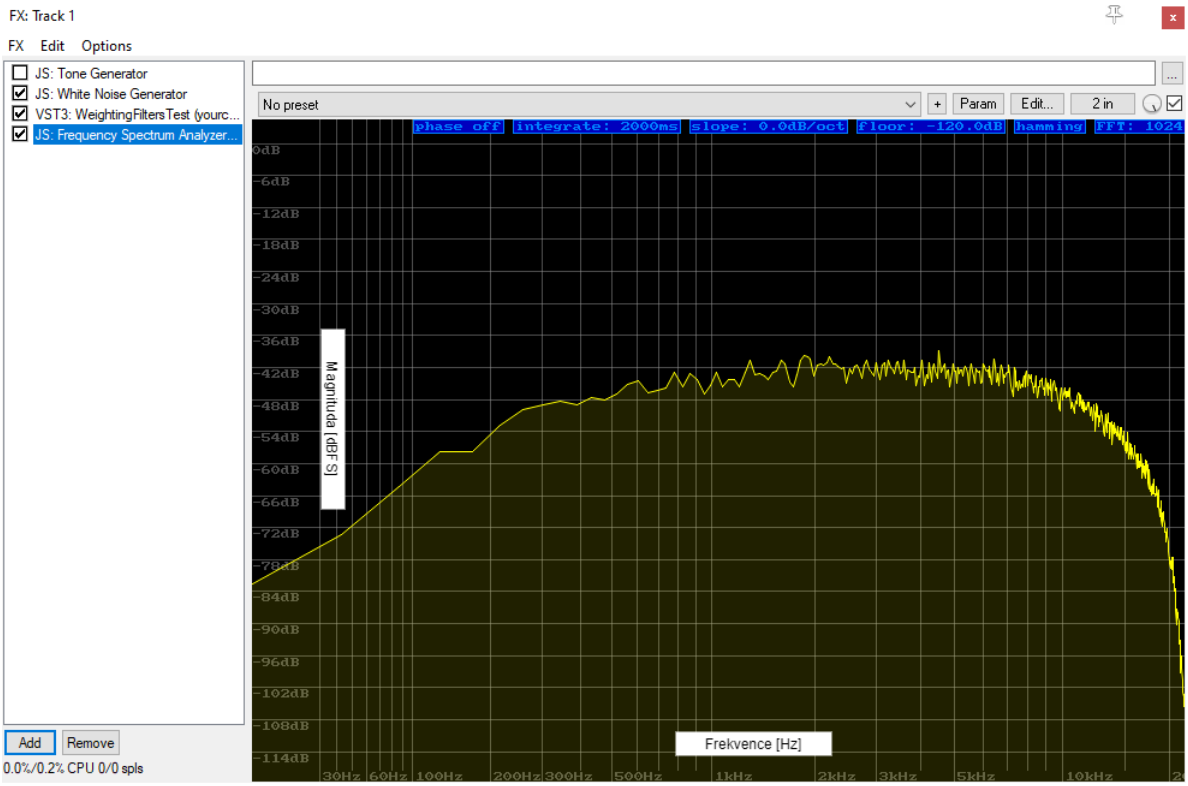
4.1, kde τ je váhovací konstanta ($\tau = 125ms$ pro váhování F, $\tau = 1s$ pro váhování S). Třída také poskytuje možnost výpočtu efektivní hodnoty namísto časového váhování.

$$f_m = \frac{1}{\tau} \tag{4.1}$$

Následně je provedeno odmocnění filtrovaného umocněného signálu, toto je provedeno pomocí cyklu `for` a funkce `sqrt` z knihovny `math.h`, jelikož abstrakci pro odmocnění celého bloku vyrovnávací pamětí pomocí SIMD instrukcí knihovna JUCE neposkytuje.

4.1.5 Implementace kalibrace zvukoměru

K realizaci kalibrace pomocí referenční hodnoty napětí či akustického tlaku (případně jejich hladin) slouží procesor `CaibrationProcessor`. Tento procesor zaobaluje instanci třídy `FrequencyWeightingProcessor` nastavenou na frekvenční váhování Z a instanci třídy `TimeWeightingProcessor` nastavenou na časové váhování s konstantou F.



Obr. 4.3: Spektrální analýza bílého šumu - Váhovací filtr A.

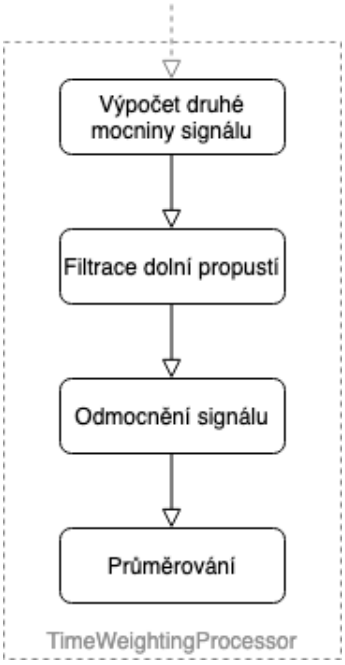
Kalibrace trvá t_{calib} sekund od spuštění, doba t_{calib} je vypočtena dle vztahu 4.2, kde τ_F je časová konstanta F (0.125 s) a t_{AVG} je doba průměrování měřených hodnot nastavená uživatelem. Konstanta τ_F je ve výpočtu času potřebného pro kalibraci použita, aby výpočet převodní konstanty nepracoval s hodnotami zahrnujícími ustálení časově váhovacího filtru.

$$t_{calib} = \tau_F + t_{AVG} \tag{4.2}$$

V případě, že při kalibraci dojde k překročení vstupního rozsahu A/D převodníku na zvukovém rozhraní, dojde k přerušení kalibrace, o kterém je uživatel informován v grafickém rozhraní. V takovémto případě je nutno snížit zisk předzesilovače na příslušném kanálu zvukového rozhraní tak, aby k překročení vstupního rozsahu nedocházelo (na zvukových rozhraních obvykle značeno indikátorem „CLIP“)

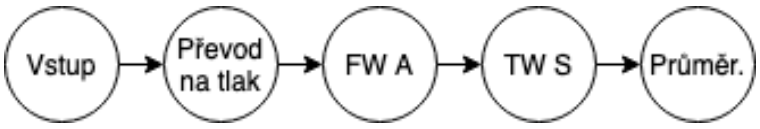
4.1.6 MeasurementProcessor

Pro propojení procesorů pro převod na tlak či napětí, frekvenční váhování a časové váhování a průměrování je použita instance třídy `juce::AudioProcessorGraph` zaobalená ve třídě `MeasurementProcessor`.



Obr. 4.4: Diagram operací nad signálem v procesoru TimeWeightingProcessor.

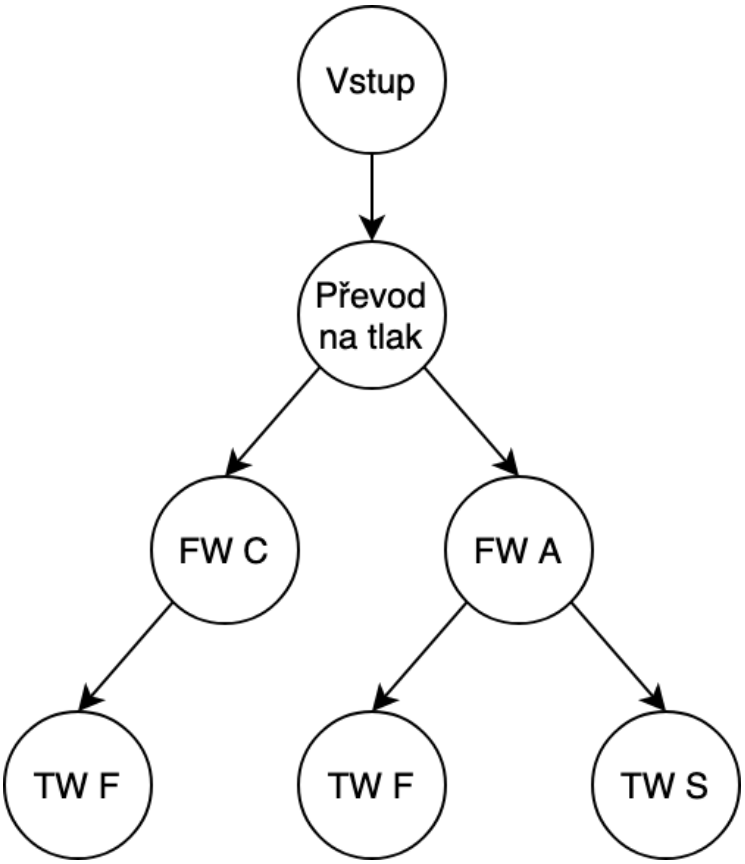
Instance třídy `MeasurementProcessor` je vytvořena v rámci hlavní komponenty `MainComponent` pro každý měřící kanál zvlášť,zprostředkovává generování a inicializaci potřebných procesorů a jejich zařazení do signálového řetězce. Grafové struktury pro dynamické přidávání a spojování bylo využito kvůli možnosti volby měření napětí či akustického tlaku se zařazením různých kombinací frekvenčních a časových váhování zároveň. Při vhodném použití lze tímto redukovat počet potřebných procesorů a sdílení instancí `ConversionProcessor`, `FrequencyWeightingProcessor` a `FrequencyWeightingProcessor` mezi více zobrazovacími prvky zvukoměru.



Obr. 4.5: Signálový řetězec pro jeden zobrazovací prvek.

Jednoduchý signálový řetězec pro měření hladiny akustického tlaku s frekvenčním váhováním A a časovým váhováním S je vyobrazen na obrázku 4.5. Při rozšíření tohoto řetězce o procesory pro další dva zobrazovací prvky, například L_{AF} s frekvenčním váhováním A a časovým váhováním F a L_{CF} s frekvenčním váhováním C a časovým váhováním F, můžeme některé procesory využít vícekrát, což je hlavní rozdíl oproti tvorbě samostatných řetězců pro jednotlivé zobrazovací prvky. Obrázek

4.6 ukazuje takto rozšířený graf signálových cest. Na obrázku 4.7 je jsou pak barevně naznačeny signálové cesty využívané pro jednotlivé měřené veličiny.



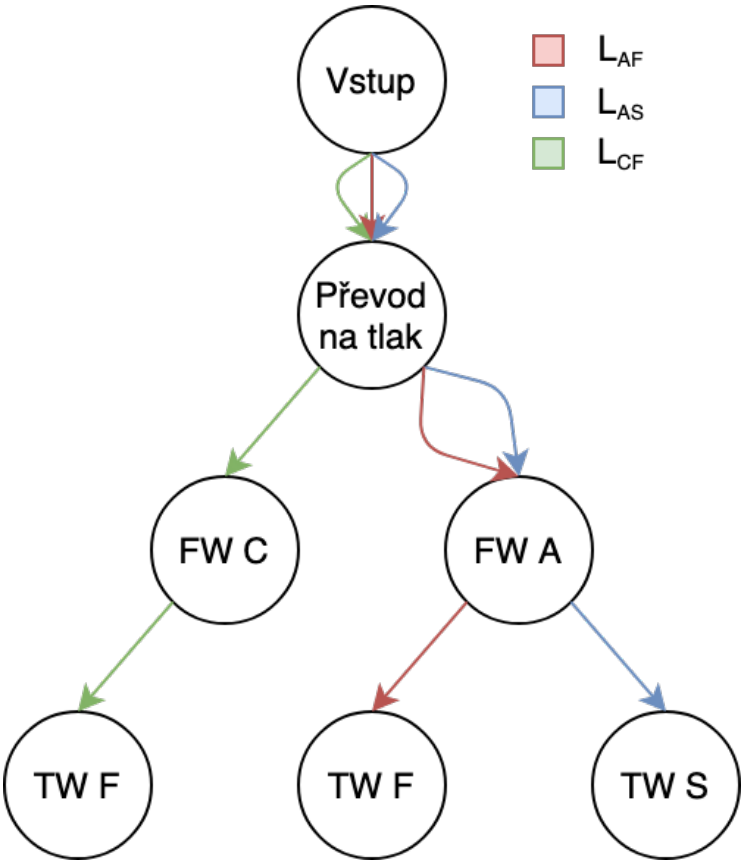
Obr. 4.6: Graf signálových cest pro vícero zobrazovacích prvků.

Postup generování procesorů a spojení mezi nimi je vyobrazen na obrázku 4.8. Zkratka „CP“ ve vývojovém diagramu značí `ConversionProcessor`, zkratka „FWP“ `FrequencyWeightingProcessor` a „TWP“ `TimeWeightingProcessor`, následované jedním písmenem značícím nastavené frekvenční či časové váhování.

4.1.7 Signálový generátor

Související metody ve zdrojovém kódu

- `updateAngleDelta`
- `updateTargetFrequencyAndLevel`
- `getNextAudioBlock`



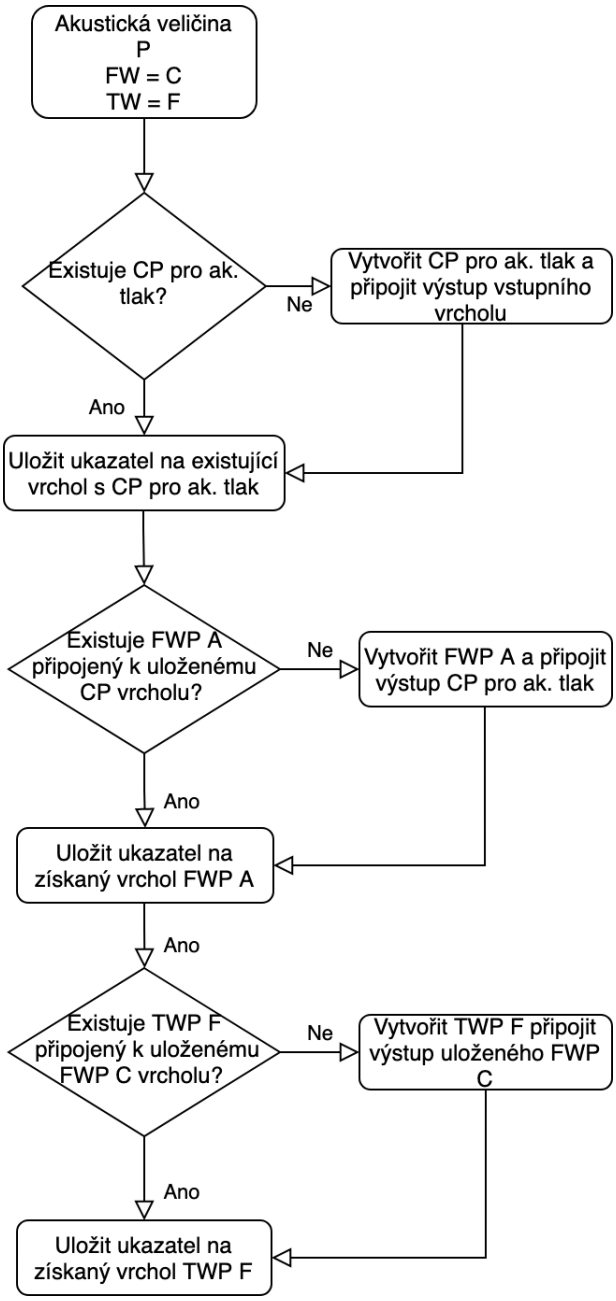
Obr. 4.7: Graf využití signálových cest pro výpočet jednotlivých veličin.

Generátor signálu je realizován pomocí cyklu `for`, jenž plní výstupní buffery dvou výstupních kanálů vypočtenými hodnotami sinusového signálu. Během každého kroku cyklu je aktuální úhel (argument pro funkci `std::sin` vytvářející jednotlivé vzorky) navýšen o úhel odpovídající počtu cyklů sinusoidy na vzorek.

Při změnách hodnot frekvence a amplitudy provedených v grafickém rozhraní jsou aktualizovány příslušené záznamy ve stromové struktuře `juce::ValueTree`. V kódu generujícím sinusový signál je pak provedena postupná změna frekvence či amplitudy, aby bylo zamezeno vzniku nespojitostí (lupání, praskání) na výstupu.

4.2 Práce s daty a stavem aplikace

Pro komunikaci mezi signálovými procesory a grafickým rozhraním je využito instance třídy `juce::ValueTree`. Instance této třídy poskytují možnost uchovávat data ve struktuře orientovaného stromu. Každá instance má svou sadu atributů („property“), do které je možno ukládat páry textového identifikátoru a hodnoty. Dále také obsahuje strukturu, do které je možno ukládat další instance třídy



Obr. 4.8: Postup generování signálových cest pro MeasurementProcessor.

`juce::ValueTree`, tedy „potomky“.

Celý stav aplikace, tedy „kořen“ stromové struktury je uchováván v členské proměnné komponenty `MainComponent` zvané `appState`, hlavní komponenta tedy má přístup ke všem datům uchovaným v této stromové struktuře. Při vytvoření komponent pro jednotlivé měřící kanály je jim předán příslušný „potomek“ náležící „kořenu“ stromové struktury. Uvnitř těchto komponent je tedy možno přistupovat pouze k nejvyššímu uzlu, který jim byl předán.

Kopírovací konstruktor třídy `juce::ValueTree` a přiřazovací operátor `operator=` provádí pouze tvorbu reference, nikoli vytvoření nové instance. Operace nad proměnnými vytvořenými přirazením či kopírováním tedy zasahují do původní instance `juce::ValueTree` a poskytují snadnou možnost synchronizace dat a stavu napříč hlubokými strukturami komponent. Při předání

Třída `juce::ValueTree` při změnách dat či struktury (atributy, potomci, změna bezprostředního předka...) posílá zprávu a poskytuje také třídu `juce::ValueTree ::Listener`, která je schopna tyto zprávy přijímat a nakládat s nimi dle implementace programátora. Zachytávané zprávy o změně jsou kromě specifikovaného uzlu také zachytávány ze všech jeho potomků. Při zpracování zprávy je mimo další informace poskytnuta reference na uzel stromové struktury, v kterém změna nastala, je tedy možno reagovat například pouze na změny ve zkoumaném uzlu, nikoli jeho potomcích, případně naopak v libovolném či specifickém potomku.

4.3 Grafické rozhraní

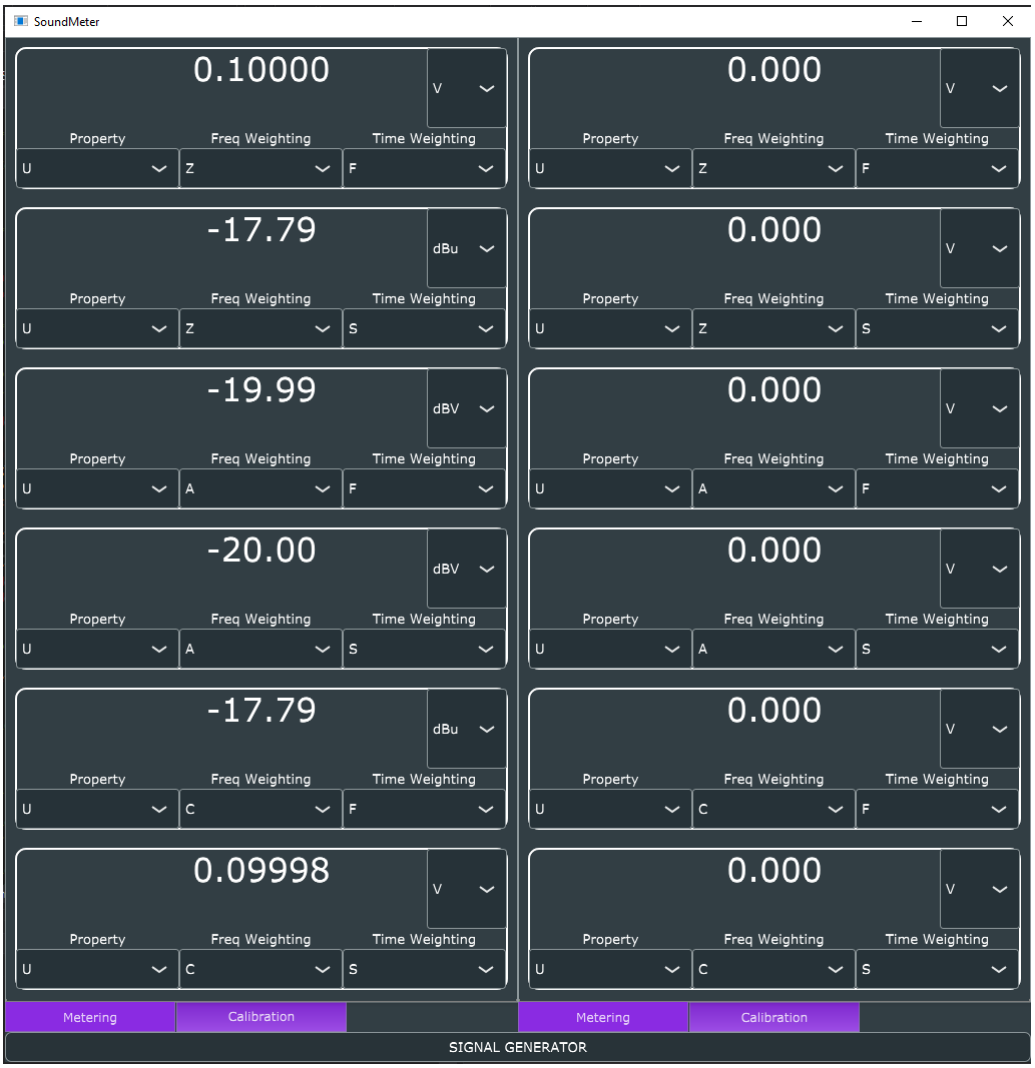
Grafické rozhraní bylo realizováno pomocí systému komponent zmíněného v 3.4. Postup zapouzdřování dílčích částí uživatelského rozhraní byl zvolen pro snadné rozšiřování a opakované použití kódu při budoucím vývoji. Komponenta `MainComponent` zahrnuje celý obsah grafického rozhraní programu SoundMeter.

4.3.1 Komponenta ChannelComponent

`ChannelComponent` v grafickém rozhraní reprezentuje jeden kanál měření. Zaobaluje záložky `MeteringTabComponent` a `CalibrationTabComponent`. Struktury `juce::ValueTree` uchovávající stav komponent ChannelComponent jsou přímými potomky kořene stromu `appState`.

4.3.2 Komponenta MeterComponent

`MeterComponent` slouží k zobrazování průměrovaných měřených hodnot vypočtených v procesorech `TimeWeightingProcessor`. Tato komponenta je zahrnuje prvky (rozbalovací seznamy) pro volbu mezi převodem na napětí či akustický tlak, dále pro volbu frekvenčního a časového váhování. Po straně je možno zvolit přepočet jednotek měřené veličiny - mV, V, dBV a dBu pro napětí a μPa, Pa, dB(SPL), dB(Pa) pro akustický tlak. Komponenty MeterComponent jsou vykreslovány přibližně 20x za sekundu díky dědění z třídy `juce::Timer` a implementaci její metody `timerCallback`. Struktury `juce::ValueTree` uchovávající stav jednotlivých komponent `MeterComponent` již nemají potomky, jsou tedy „listy“ stromu.



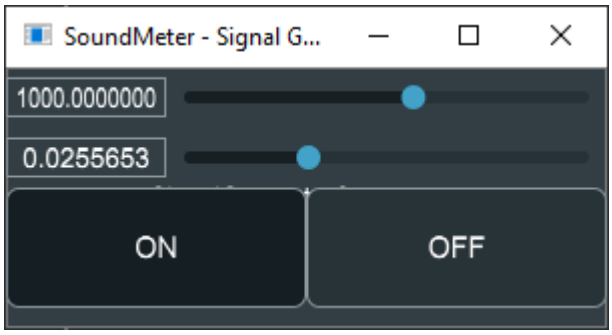
Obr. 4.9: Hlavní okno programu SoundMeter.

4.3.3 Komponenta SignalGeneratorComponent

`SignalGeneratorComponent` slouží k ovládání generátoru sinusového signálu na výstup zvukového rozhraní. Obsahuje dva posuvné prvky realizované pomocí komponenty `Slider` a přepínač realizovaný pomocí komponent `TextButton`. První z posuvných vstupních prvků slouží k zadání frekvence generovaného sinusového signálu, druhý k nastavení amplitudy tohoto signálu.

Hodnota zesílení signálu je implicitně nastavena na -INF dB (pro účely tohoto programu ticho na výstupu) pro snížení rizika případného poškození vybavení či sluchu, pokud by uživatel předem nezkontroloval nastavení zesílení výstupu zvukového rozhraní či prvků následujících za ním.

Ovládací prvek pro frekvenci je pomocí metody `setSkewFactorFromMidPoint`



Obr. 4.10: Okno signálového generátoru v programu SoundMeter.

nastaven na nelineární průběh, kde prostřední hodnota posuvníku je 500 Hz. Toto slouží k docílení vyššího rozlišení posuvníku na nižších frekvencích. Oběma ovládacím prvkům je možno zadat hodnotu v číselném formátu, v políčku po jejich levé straně.

Ačkoli komponenta náleží kontextu komponenty `MainComponent` (její stav je uchováván po celý běh programu), je zobrazována v samostatném okně (komponenta `SignalGeneratorWindow`). To je vytvořeno po stisknutí tlačítka "Signal Generator", opakované stisknutí okno vyvolá do popředí. Pomocí metody `SignalGeneratorWindow::setContentNotOwned` je jako obsah okna nastavena komponenta signálového generátoru, její stav však bude zachován i po zavření tohoto okna. Zavření okna je třeba provést pomocí tlačítka pro zavření okna poskytnutého operačním systémem.

4.4 Ověření funkčnosti programu

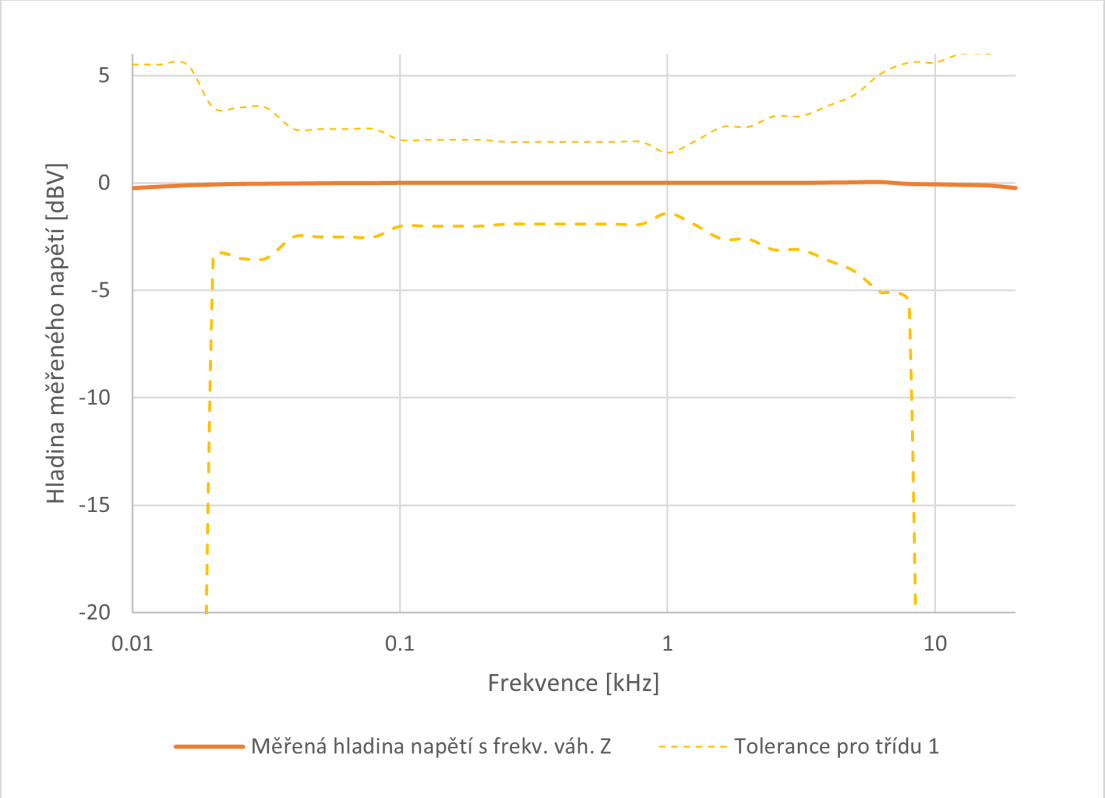
Program byl otestován jednak samostatně za pomoci generátoru audio signálů NTi Minirator MR2 a také ve srovnání s profesionálním zvukoměrem NTi XL2 s mikrofonem NTi M4261.

4.4.1 Funkce měření napětí

Pro ověření funkce měření napětí byl použit generátor audio signálu NTi Minirator MR2. Toto měření bylo provedeno za pomoci cenově dostupného zvukového rozhraní Behringer U-Phoria UMC404HD. Po zapojení signálového generátoru do prvního kanálu zvukového rozhraní pomocí kabelu XLR-XLR byl na generátoru nastaven tvar vlny na „SINE“ (sinus), výstupní napětí na 0 dBV a frekvence na 1 kHz.

Na zvukovém rozhraní byl pomocí tlačítka „PAD“ zařazen tlumící článek poskytující utlumení vstupního signálu přibližně o 20 dB, aby bylo možno zkalibrovat

softwarový zvukoměr pomocí napětí 0 dBV pro snazší vyhodnocování měření. Bez zařazení tlumícího článku by takto vysoké napětí způsobilo přetížení předzesilovače zvukového rozhraní, případně A/D převodníku a tedy znehodnotilo kalibraci i samotné měření.

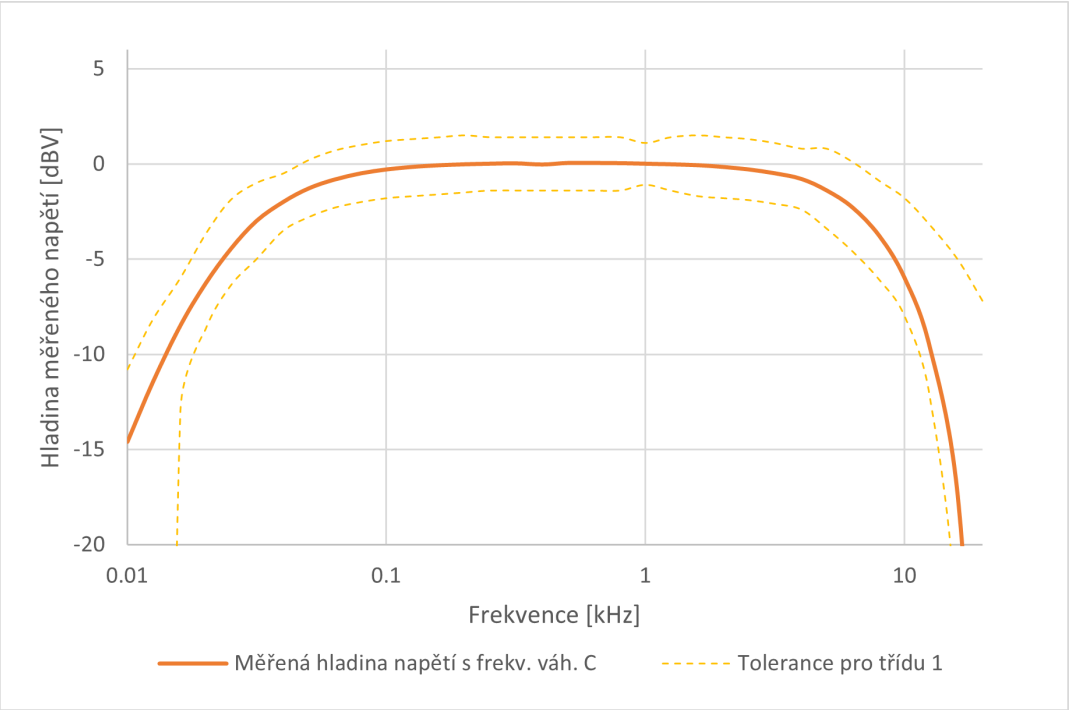


Obr. 4.11: Výsledek měření hladiny napětí s frekvenčním váhováním Z.

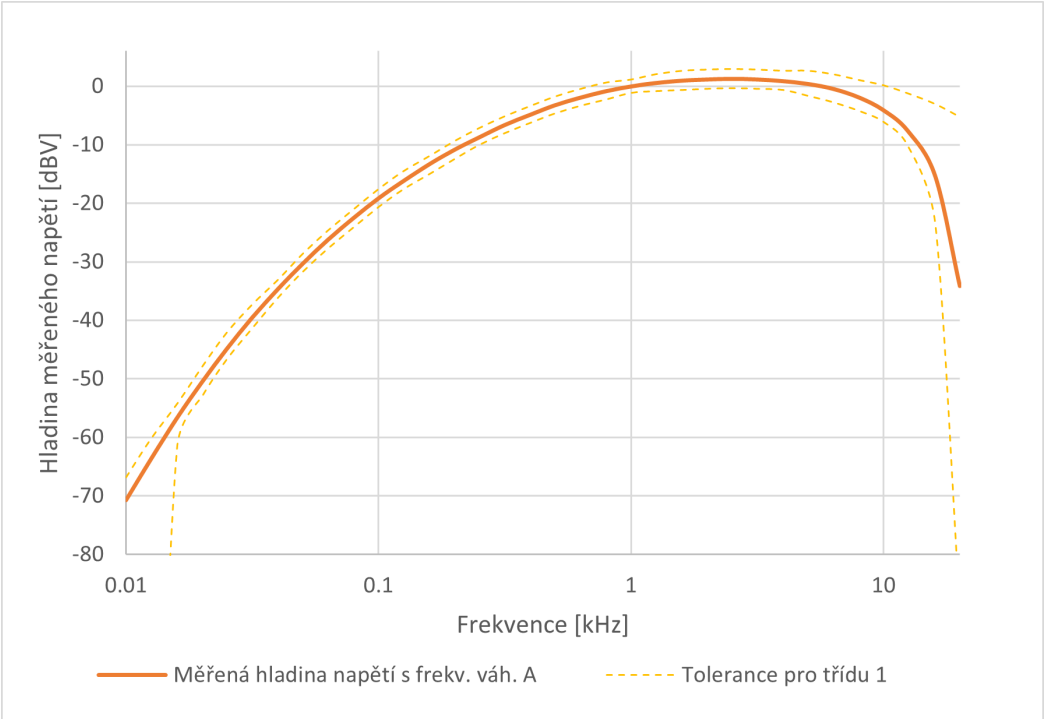
Při tomto měření byla kromě funkce kalibrace ověřována také kombinovaná frekvenční odezva váhovacích filtrů a signálové cesty zvukového rozhraní. Měření pro všechna tři frekvenční váhování bylo provedeno se zařazeným časovým váhováním F a třísekundovým plovoucím průměrováním. Výsledky měření jsou uvedeny v spolu s tolerančním rozsahem pro zvukoměry třídy 1 dle IEC 61672. Výsledek měření pro frekvenční váhování Z je uveden v grafu 4.11, pro frekvenční váhování C v grafu 4.12 a pro frekvenční váhování A v grafu 4.13. [2]

4.4.2 Funkce měření akustického tlaku

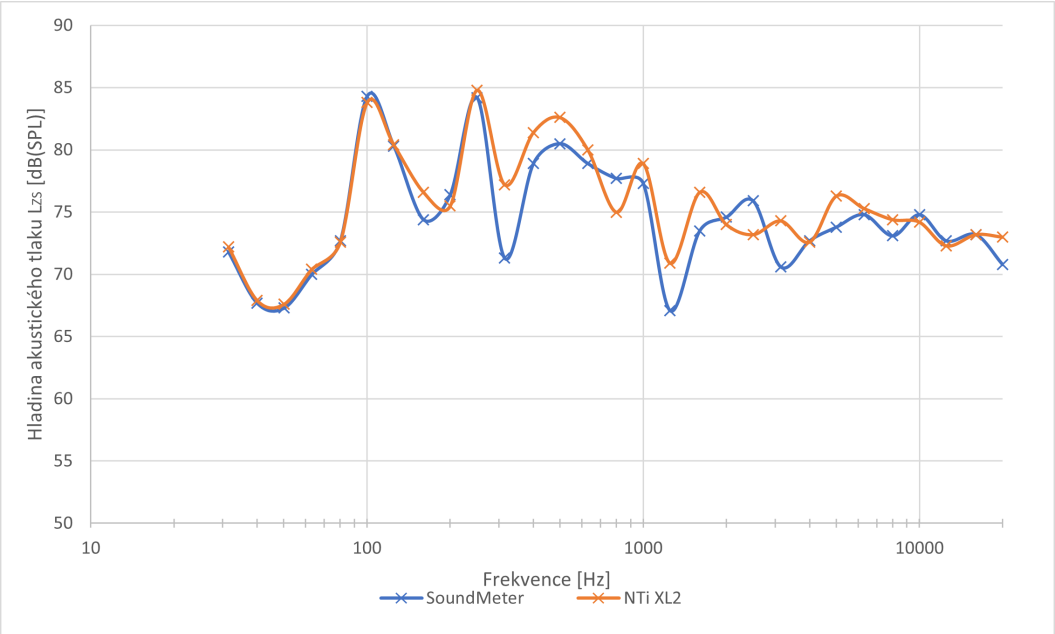
Správné fungování měření akustického tlaku bylo ověřováno za pomoci generátoru NTi Minirator MR2, zvukoměru NTi XL2 a měřícího mikrofonu NTi M4261. Jako zdroj zvuku byla použita aktivní studiová reprosoustava značky Event, na jejíž vstup byl připojen generátor signálu.



Obr. 4.12: Výsledek měření hladiny napětí s frekvenčním váhováním C.



Obr. 4.13: Výsledek měření hladiny napětí s frekvenčním váhováním A.

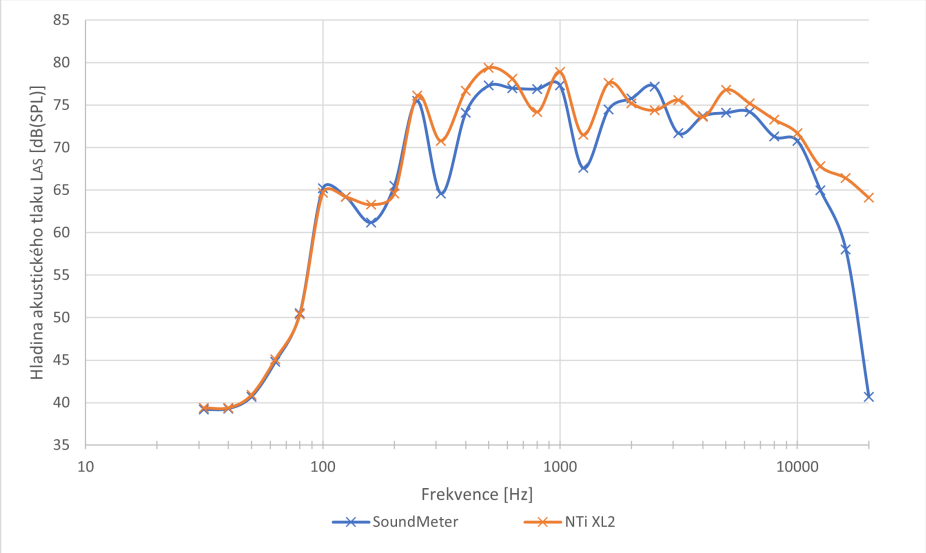


Obr. 4.14: Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování Z).

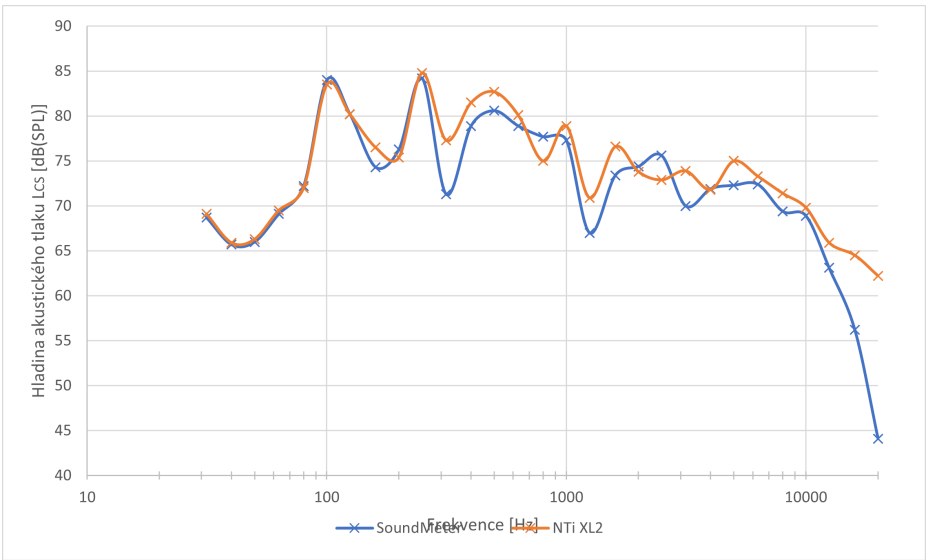
Přibližně 1 m od reprosoustavy byl umístěn měřící mikrofon na stojanu, jenž byl připojen ke zvukoměru XL2. Na zvukoměru XL2 byla provedena kalibrace za pomoci kalibrátoru generujícího akustický tlak 1 Pa při frekvenci 1 kHz. Následně bylo provedeno měření hladiny akustického tlaku při nastavení frekvence na generátoru v rozsahu 31,5 Hz až 20 kHz s třetinooktávovým krokem.

Kalibrace a měření bylo provedeno také za použití programu SoundMeter se zvukovým rozhraním Steinberg UR44. Uvedené výsledky jsou měřeny se zařazeným časovým váhováním s konstantou S. Srovnání výsledků měření je provedeno v grafech 4.14 pro frekvenční váhování Z, 4.15 pro frekvenční váhování A a 4.16 pro frekvenční váhování C.

Zvukoměr XL2 má výrazně kratší dobu klouzavého průměrování než 3 sekundy v případě programu SoundMeter, měření pomocí něj bylo tedy náchylnější na nahodilé ruchy či zvuk manipulace se zařízeními v místnosti. To je jedním z možných vysvětlení odchylek o jednotky dB v celém frekvenčním rozsahu. Z porovnání měřených hodnot u frekvenčního váhování A a C je však zřejmé, že zvukoměr XL2 používá odlišně sestrojený váhovací filtr. Program SoundMeter má dolní propust váhování A a C výrazně strmější, než zvukoměr XL2 (ačkoli při měření napětí spadá do tolerancí pro zvukoměry třídy 1).



Obr. 4.15: Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování A).



Obr. 4.16: Srovnání měřených hodnot zvukoměry NTi XL2 a SoundMeter (frekvenční váhování C).

Závěr

Tato práce se zabývala problematikou měření zvukových veličin, digitálním zpracováním signálu a návrhem dílčích částí programu určeného pro měření napětí, akustického tlaku a jejich hladin.

V první části této práce byly popsány vybrané akustické veličiny, základní principy digitálního zpracování signálu a zařízení potřebné k realizaci softwarového zvukoměru.

Druhá část této práce se zabývala popisem zařízení zvukoměru, jeho součástí a požadavků kladených standardem IEC 61672.

Ve třetí části byly popsány softwarové nástroje, především jazyk C++ a framework JUCE, využité při vývoji a testování programu SoundMeter.

V poslední části byl popsán postup návrhu a implementace programu SoundMeter, jeho tříd a funkcí a grafického rozhraní. Zdrojový kód programu je k dispozici v příloze, včetně spustitelné verze pro Windows.

Program SoundMeter poskytuje možnost kalibrace na specifikované napětí či akustický tlak a následného dvoukanálového měření se zařazením frekvenčního váhování, časového váhování a klouzavého průměrování. Signálový generátor byl implementován (třída `SignalGeneratorProcessor`), jeho napojení na příslušné komponenty uživatelského rozhraní pro kalibraci a ovládání v dosavadní verzi nebylo úspěšné.

Na výsledky této práce by bylo možno navázat implementací dalších funkcí zvukoměru, například měření ekvivalentní hladiny akustického tlaku, minim a maxim...

Knihovny pro kalibraci, převod z digitální hodnoty na napětí či akustický tlak, frekvenční a časové váhování, by do budoucna mohly být po mírné úpravě základem pro jiný software či projekt s nenáročným hardware (například Raspberry Pi) zpracovaný na základě knihovny JUCE, například instalační zvukoměr připojený k internetu pro sběr dat k hlukovým studiím, ať už environmentálním, či průmyslovým.

Literatura

[1] LERCH, A. *An Introduction to Audio Content Analysis*. Hoboken, NJ: Wiley Publishing, 2012. 259 s. ISBN 978-1-118-26682-3.

[2] *61672 Electroacoustics - Sound level meters.* 40 stran. Ženeva: International Electrotechnical Commission, 2002.

[3] EARGLE, J. *The Microphone Book*. Burlington, MA: Focal Press, 2005. 377 s. ISBN 02405 1961 2.

[4] GINSBERG, J. H. *Acoustics A Textbook for Engineers and Physicists*. Cham: Springer, 2018. 596 s. ISBN 978-3-319-56844-7.

[5] SCHIMMEL, J. *Elektroakustika*. Brno: Vysoké učení technické v Brně, 2013. ISBN: 978-80-214-4716-5.

[6] JUCE. *JUCE* [online]. Dostupné z: `https://juce.com/`

[7] JUCE: Class index. *JUCE* [online]. Dostupné z: `https://docs.juce.com/master/index.html`

[8] Tutorial: Build a sine wave synthesiser. *JUCE* [online]. Dostupné z: `https://docs.juce.com/master/tutorial_sine_synth.html`

[9] Tutorial: Create a basic Audio/MIDI plugin, Part 2: Coding your plug-in. *JUCE* [online]. Dostupné z: `https://docs.juce.com/master/tutorial_code_basic_plugin.html`

[10] Tutorial: Cascading plug-in effects. *JUCE* [online]. Dostupné z: `https://docs.juce.com/master/tutorial_audio_processor_graph.html`

[11] File:FletcherMunson ELC.png. *Wikimedia Commons* [online]. Dostupné z: `https://commons.wikimedia.org/wiki/File:FletcherMunson_ELC.png`

[12] File:SVAN 979 Sound Level Meter and Analyzer.jpg. *Behringer* [online]. Dostupné z: `https://www.behringer.com/product.html?modelCode=P0118`

[13] File:SVAN 979 Sound Level Meter and Analyzer.jpg. *Wikimedia Commons* [online]. Dostupné z: `https://commons.wikimedia.org/wiki/File:SVAN_979_Sound_Level_Meter_and_Analyzer.jpg`

[14] PRYOR, Sam. SYSTEM SCIENCE - PART 2: DRIVERS & LATENCY. *The Focusrite group* [online]. Dostupné z: `https://focusrite.com/en/news/system-science-part-2-drivers-latency`

[15] Measuring Sound with Microphones. *National Instruments* [online]. [cit. 2020-12-11]. Dostupné z: https://www.ni.com/cs-cz/innovations/white-papers/13/measuring-sound-with-microphones.html

[16] Measurement Microphones Explained. *BRÜEL & KJÆR* [online]. [cit. 2020-12-11]. Dostupné z: https://www.bksv.com/en/Knowledge-center/blog/articles/sound/measurement-microphones

[17] LEVINE, Mike. Measurement Microphones Explained. *Yamaha* [online]. [cit. 2020-12-11]. Dostupné z: https://hub.yamaha.com/what-is-an-audio-interface/

[18] Unattended Noise Monitoring. *NTi Audio* [online]. [cit. 2020-12-11]. Dostupné z: https://www.nti-audio.com/en/applications/noise-measurement/unattended-monitoring

[19] Nařízení vlády č. 9/2002 Sb. *Zákony pro lidi* [online]. [cit. 2020-12-11]. Dostupné z: https://www.zakonyprolidi.cz/cs/2002-9

[20] Zákon č. 258/2000. *Zákony.cz* [online]. 2000 [cit. 2020-12-11]. Dostupné z: urlhttps://www.zakony.cz/zakony/2000/201/zakon-258-2000-Sb-SB2000258

Seznam symbolů a zkratek

Šířka levého sloupce Seznamu symbolů, veličin a zkratek je určena šířkou parametru prostředí `acronym` (viz řádek 1 výpisu zdrojáku na str. **??**)

KolikMista pouze ukázka vyhrazeného místa

DSP číslicové zpracování signálů – Digital Signal Processing

f_{vz} vzorkovací kmitočet

A Obsah přiloženého archivu

Program byl sestaven a testován pomocí vývojových prostředí Visual Studio 2019 pod systémem Windows 10 a Xcode pod systémem macOS.

```
/
├── logo
├── Builds ..... sestavení programu a pomocné soubory VS2019
├── JuceLibraryCode ..... kód knihovny JUCE
├── Source ..... zdrojový kód programu SoundMeter
├── SoundMeter.jucer ..... pomocný soubor Projucer projektu
└── .gitignore .....Pomocný soubor verzovacího systému Git
```